Md Kayesh

JS

Four Principles of Object-Oriented Programming

OOP

Md Kayesh

JS

1. Encapsulation

Even if we have no idea what encapsulation is, we can guess just by the word that we are “enclosing” or hiding something, that’s exactly what is! With encapsulation, object’s methods and properties are enclosed within the object, so there are not exposed.

```
class Person {
  constructor(firstName, lastName, dateOfBirth) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.dateOfBirth = dateOfBirth;
  }

  getFullName() {
    return `${this.firstName} ${this.lastName}`;
  }

  getAge() {
    return new Date().getFullYear() - this.dateOfBirth;
  }
}

const john = new Person("John", "Doe", 1990);

console.log(john.getFullName()); // John Doe;
console.log(john.getAge()); // 34;
console.log(john.firstName); // John;
```

Md Kayesh

JS

2.Abstraction

If we look at the car, we can only see what is outside. We see wheels, doors, windows and so on. But what we don't see is the complex engine and all those inner parts. That's what abstraction is: hiding complex details and showing simple ones. It helps the code to be more understandable.

```
class MethodsOfArray {
  constructor(array) {
    this.array = array;
  }

  getArray() {
    return this.array;
  }

  getSum() {
    return this.array.reduce((a, b) => a + b);
  }

  getAverage() {
    return this.getSum() / this.array.length;
  }
}

const arr = new MethodsOfArray([1, 2, 3, 4, 5]);

console.log(arr.getArray()); // [1, 2, 3, 4, 5];
console.log(arr.getSum()); // 12;
console.log(arr.getAverage()); // 2.4
```

Md Kayesh

JS

3.Inheritance

Inheritance allows for parent class to pass functionality to a child class, again, creating clean and reusable code, avoiding repeats. If we have class Player that has name, dateOfBirth and function 'getAge' and class FootballPlayer with the same function, we can extend that function into the FootballPlayer. Here is how it looks in code.

```
class Player {
  constructor(name, dateOfBirth) {
    this.name = name;
    this.dateOfBirth = dateOfBirth;
  }

  getAge() {
    return new Date().getFullYear() - this.dateOfBirth;
  }
}

class FootballPlayer extends Player {
  constructor(name, dateOfBirth, goals) {
    super(name, dateOfBirth);
    this.goals = goals;
  }
}

const Messi = new FootballPlayer("Leo Messi", 1985, 550);

console.log(Messi.getAge()); // 39
```

Md Kayesh

JS

4. Polymorphism

We think of polymorphism as something having multiple forms. In this instance, polymorphism means the same method can be used on different objects. For example , if a Bird and a Panguin have the same function — fly, polymorphism gives us an ability to call the same method on different objects.

```
class Bird {
  constructor(name) {
    this.name = name;
  }

  fly() {
    console.log(`${this.name} is flying.`);
  }
}

class Penguin extends Bird {
  fly() {
    console.log(`${this.name} can't fly.`);
  }
}

const bird = new Bird("Sparrow");
const penguin = new Penguin("Pinguin");

bird.fly(); // Output: Sparrow is flying.
penguin.fly(); // Output: Pingu can't fly.
```

Follow for more programming lips and tricks like this

Md Kayesh